SDN Enhanced Resource Orchestration of Containerized Edge Applications for Industrial IoT

Okwuibe, Jude; Haavisto, Juuso; Harjula, Erkki; Ahmad, Ijaz; Ylianttila, Mika

Published in:
IEEE Access

DOI:
10.1109/ACCESS.2020.3045563

Published: 01/12/2020

Document Version
Publisher's final version

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 26, 2020, accepted December 13, 2020, date of publication December 17, 2020,
date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3045563

SDN Enhanced Resource Orchestration of Containerized Edge Applications for Industrial IoT

JUDE OKWUIBE[1], JUUSO HAAVISTO[2], ERKKI HARJULA[1], (Member, IEEE), IJAZ AHMAD[3], (Member, IEEE), AND MIKA YLIANTTILA[1], (Senior Member, IEEE)

[1]Center for Wireless Communication, University of Oulu, 90014 Oulu, Finland
[2]Center for Ubiquitous Computing, University of Oulu, 90014 Oulu, Finland
[3]VTT Technical Research Center of Finland, 02044 Espoo, Finland

Corresponding author: Jude Okwuibe (jude.okwuibe@oulu.fi)

This work was supported in part by the Academy of Finland 6Genesis Flagship under Grant 318927, and in part by the AI Enhanced Mobile Edge Computing project, funded by the Future Makers program of Jane and Aatos Erkko Foundation and Technology Industries of Finland Centennial Foundation. The work of Ijaz Ahmad was supported by the Jorma Ollila Grant.

ABSTRACT With the rise of the Industrial Internet of Things (IIoT), there is an intense pressure on resource and performance optimization leveraging on existing technologies, such as Software Defined Networking (SDN), edge computing, and container orchestration. Industry 4.0 emphasizes the importance of lean and efficient operations for sustainable manufacturing. Achieving this goal would require engineers to consider all layers of the system, from hardware to software, and optimizing for resource efficiency at all levels. This emphasizes the need for container-based virtualization tools such as Docker and Kubernetes, offering Platform as a Service (PaaS), while simultaneously leveraging on edge technologies to reduce related latencies. For network management, SDN is poised to offer a cost-effective and dynamic scalability solution by customizing packet handling for various edge applications and services. In this paper, we investigate the energy and latency trade-offs involved in combining these technologies for industrial applications. As a use case, we emulate a 3D-drone-based monitoring system aimed at providing real-time visual monitoring of industrial automation. We compare a native implementation to a containerized implementation where video processing is orchestrated while streaming is handled by an external UE representing the IIoT device. We compare these two scenarios for energy utilization, latency, and responsiveness. Our test results show that only roughly 16 percent of the total power consumption happens on the mobile node when orchestrated. Virtualization adds up about 4.5 percent of the total power consumption while the latency difference between the two approaches becomes negligible after the streaming session is initialized.

INDEX TERMS 5G, AR, containerization, cloud, Docker, edge computing, HWPC, industry 4.0, InfluxDB, IoT, IIoT, latency, MongoDB, multi-access, MNO, NFV, PowerAPI, power consumption, software defined networking (SDN), VR.

I. INTRODUCTION

The ongoing evolution in industrial automation and connectivity for smart factories has created an avenue for harnessing new and existing technologies towards the improvement of productivity and efficiency across manufacturing and process automation. Efforts in both academia and the industry are geared towards streamlining business operations and manufacturing processes to meet the high expectations of the ongoing industrial revolution; the industry 4.0 [1]. This evolution cuts across various industry sectors including supply chain management, monitoring systems, data analytics, and various feedback loops. The Industrial Internet of Things (IIoT) is one of such moves where IoT is augmented with intelligent and big data analytics for enhancing industrial processes [2]. Unlike with IoT systems where certain margin for error could be tolerable, with IIoT, the margin for errors is extremely low, given the high level of sensitivity and precision with which such operations and processes must be handled. IIoT is therefore characterized by massive interconnection of devices, location-awareness, sophisticated advanced controls and analytics, availability, and intelligent systems [3].

Another key defining element for IIoT is the massive amount of data being generated and transmitted across

The associate editor coordinating the review of this manuscript and approving it for publication was Chun-Wei Tsai.

interconnected devices. These include cameras, actuators, transducers, and other data-intensive multimedia sensors. These technologies are harmonized to provide real-time data on the state of the devices as well as the state of the industrial processes supported by such devices. In addition, most IIoT use cases require these devices to be location aware in order to enhance accuracy, safety, and effectiveness. With such an amount of data and interconnected devices comes the need for a dynamic connection management solution, cloud integration, machine learning, as well as some advanced data offloading techniques.

Software Defined Networking (SDN) leverages on the separation of the control and data planes from underlying routers and switches to offer a more dynamic approach to networking through logically centralized programmable control functions in IP networks [4]. This change in paradigm simplifies various network functions such as policy enforcement, network reconfiguration and programmability. With these features, leveraging SDN for networking in the industrial automation domain becomes the natural trajectory for the ongoing and future industrial revolution. More specifically, in areas such as industrial Ethernet [5]–[7] and wireless technologies [8], [9]. In these areas, the flexibility and programmability of SDN, coupled with its ability to host the control logic in an external network component called a controller, underscores the need for SDN in interconnecting IIoT devices.

Container technologies such as Docker and Kubernetes are fast becoming the de-facto approach for implementing operating system level virtualization for delay-sensitive IoT applications. On one hand, this is a consequence of their lightweight architecture, which enables less resource utilization and low bootstrapping time. On the other hand, their technical compliance with various deployment platforms, such as Virtual Machines (VMs), Physical Machines (PMs), and cloud environments makes the adoption easier in both the academia and the industries [10]. Primarily, containers are designed to provide a standardized isolation platform for application deployment which allows developers to isolate their applications from the environment. This provides a working solution for the long-standing platform dependency problem which hitherto was a major issue for most deployment environments. Moreover this simplification and speeding up of container deployment and configuration process also implies a corresponding ease for undeploying a deployed container. This comes in handy for situations where you are launching a product and you cannot foresee the size of the traffic it will create. In such a situation, you can leverage this flexibility factor by cloning more containers to handle growing traffic or destroying unused containers to save cost on cloud environment [11]. No wonder companies like Google, Slack, Spotify, Shopify, Pinterest, eBay, and Twitter are already using container technologies for efficient scaling of their services. This basic factor underscores the impetus behind leveraging containers for IoT service orchestration and the benefits they hold for 5G and future networks.

Multi-Access Edge Computing (MEC) services in 5G and future networks are envisioned to run as close to the User Equipment (UE) as possible through a series of optimization techniques. In practice, these applications can be considered as agents which move within the Mobile Network Operator (MNO)'s Radio Access Network (RAN), reserving resources and migrating from one MEC host to another in a dynamic fashion. On the MEC hosts, this functionality can be achieved with an orchestrator. The orchestrator's purpose is to enable resource scheduling and application migration while integrating with various parts of the RAN. However, this sort of mobile and distributed operation infrastructure poses challenges in software engineering. Considering end-user edge applications running on the MNO's infrastructure, the software has to be designed for mobility. In other words, the developer has to assume the host process can and will migrate between MEC hosts. The two common reasons for such migration are to preserve latency guarantees and to optimize radio resources with mobile UEs.

In software engineering, similar challenges have been studied in the context of microservices, which operate in containers managed by an orchestrator. Microservices are documented to address scaling challenges of business components and agile software development with large teams of engineers. Furthermore, by limiting and agreeing on interfaces and on how software components are deployed, e.g., via containers, the operational infrastructure is further simplified. This facilitates maintainability, and system reliability as both the hardware in the datacenter and the software managed by an orchestrator can be mutated as long as it adheres to container runtime specifications.

Now, with 5G, we are introduced to a shift towards SDN and Network Function Virtualization (NFV), which marks an evolution from high Capital Expenditure (CapEx) digital signal processor hardware towards more interoperable, general-purpose computing-based cellular network architecture. With these changes, the MNOs are offered with ever more affordable means of innovation in offering site-specific services accessible on UEs. That is because the SDN hardware of the teleoperator can double as general-purpose computing hardware. It is envisioned that some of these computation capabilities could be passed on to the UEs via MEC. In literature, the MEC paradigm is often referred to as the enabler for Augmented Reality (AR), Virtual Reality (VR), autonomous cars, and inter-MNO resource market called network slicing.

In all ramifications, IIoT service orchestration in containerized edge applications offers an unparalleled advantage at various levels of implementation. The ease of implementation, flexibility, scalability, component re-use, and public sharing has gained a lot of attention. However, there is no much research on the trade-offs that come with such architectural modifications. One way to bring this to light is by examining these trade-offs in the light of the traditional modes of implementation.

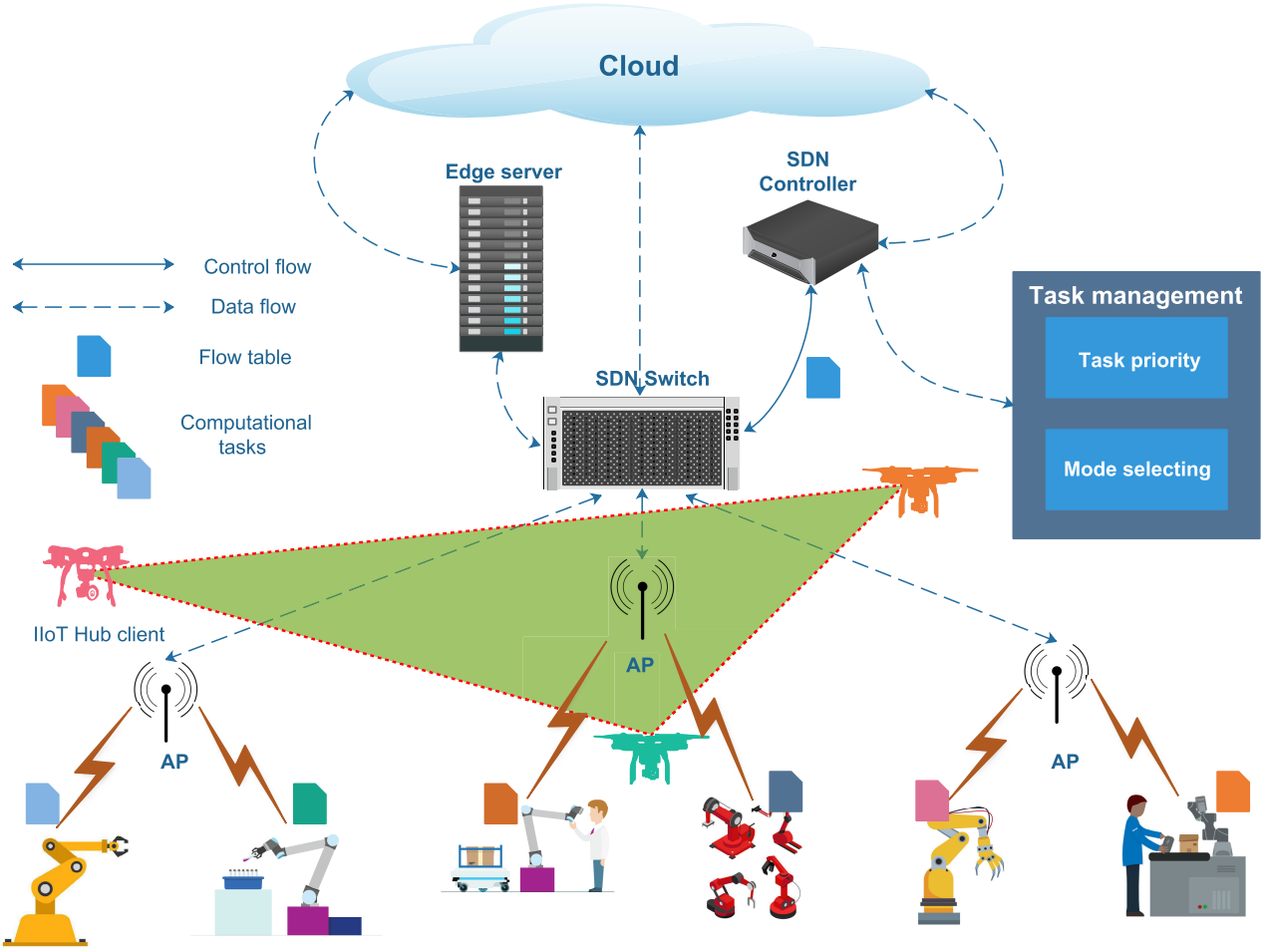

FIGURE 1. Software defined IIoT model.

To the best of the authors' knowledge, this is the first work that provides a quantitative analysis of the power and latency tread-offs for bootstrapping SDN, MEC and containerization techniques in enhancing IIoT applications.

The main contributions are summarized as follows:

- We present our vision on how to effectively leverage the synergy between SDN, MEC, and container technologies in advancing IIoT applications for better resource management.
- We propose an offloading technique that offers efficient and scalable resource management for containerized edge microservices. Here we focus on minimizing the resource utilization for edge co-located IIoT services and applications by offloading processes to a more capable platform on the edge.
- We present a performance analysis of the proposed integration based on latency and power utilization. We compare the container-based implementation to a native implementation while considering the effects of the I/O device capabilities. This quantitative analysis will serve as a good reference for further optimization of the proposed integration for both IoT and IIoT use cases.
- We further present a use case of our prototype for a 3D-drone-based monitoring system capable of providing real-time visual monitoring for industrial IoT operations.

The rest of this paper is organized as follows: Section II presents the state of the art related to IIoT, SDN-based IIoT solutions as well as MEC-enabled IIoT application areas. Section III provides an overview of the system framework of the proposed solution. This includes the server and client application designs, the service migration techniques, as well as the environment and system framework on which empirical observations and data collections are done. Section IV describes the power measurement approach used on both the server and the IIoT device. Section V provides the measurement results and evaluates the reactive system on an Over-the-Air (OTA) environment on Long Term Evolution (LTE) network. Section VI provides a discussion on the subject matter based on our results and other related works. It also proffers a future direction for the research work. Finally, the contributions are concluded in Section VII.

II. RELATED WORK

A. INDUSTRIAL INTERNET OF THINGS

The initial phases of industrial revolutions saw a transition from complete human labor to digitalization of industrial processes and operations [12]–[16]. The ongoing

TABLE 1. Comparing IoT and IIoT.

	Consumer IoT	Industrial IoT
Focus areas	Households, SMEs, customer-oriented operations and services, products, smart city	Manufacturing, facilities, supply chain monitoring, management systems, process automation, internal optimizations
Ecosystem architecture	Public cloud accessible by the ecosystem operator, Azure IoT edge	Private cloud operated by service provider
Sophistication	Scalable platform, simplified solutions, out of the box implementation	Machine-to-machine (M2M) communication, advanced controls and analytics
Sensitivity	More fault tolerant, less sensitive (except for medical applications)	Mission critical, more sensitive and precise sensors, self-monitoring, machine learning, location awareness
Degree of application	Simpler applications, low risk impacts	More complex applications, high risk impacts
Data management	Cloud-based processes, medium to high data volumes	Secure cloud, big data, high to very high data volumes
Connectivity	Ad-Hoc (infrastructure is not tolerated: nodes can be mobile)	Structured (nodes are fixed: centralized network management) [22]

industrial revolution; the industry 4.0 leverages modern smart technologies to automate traditional manufacturing and industrial practices. This revolution is mainly characterized by digital transformation, autonomous systems, IoT, Artificial Intelligence (AI), self-monitoring, and smart manufacturing [17].

The term Internet of Things refers to a network of physical objects embedded with sensors, programs, and other related technologies for the purpose of exchanging data over the internet to inform predefined actions. This definition mostly applies to connected devices in consumer, domestic, and business settings. Although there is a significant similarity with what later became the Industrial Internet of Things (IIoT), this definition does not fully represent the key elements of IIoT. This includes sensitivity, precision, location-awareness, security, sophistication and very significantly; massive amount of data generated. As such, the adoption of IoT technologies in Industrial Automation and Control Systems (IACS) needed a separate categorization, hence the term IIoT [18]. Table 1 presents a more concise comparison between consumer IoT and Industrial IoT.

In [18], authors present an analytical framework for IIoT in relationships to cyber-physical systems and Industry 4.0. This framework enumerates and characterizes IIoT devices for system architecture studies and also for analysing security threats and vulnerabilities.

Gilchrist *et al*. [19] explores the potential for IoT, Big Data, Cyber-Physical Systems (CPS), and Smart Factory technologies to replace the still largely mechanical, people-based systems of offshore locations. Here, authors analyzed trends and technologies related to IIoT and Industry 4.0. They further presented the state of the art and proffered on what technologies must advance to enable Industry 4.0. Some examples of the implementation of Industry 4.0 were also presented along with potential leads and possible fallout that could result.

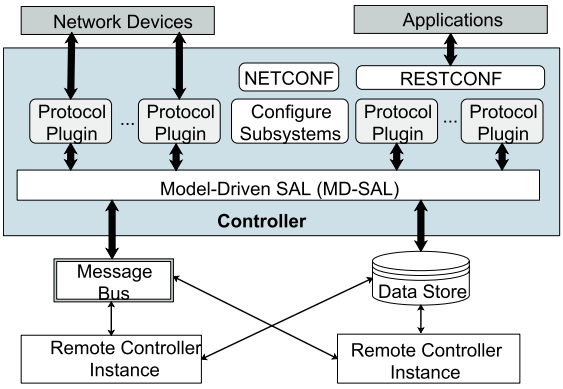

FIGURE 2. OpenDaylight controller architecture.

Wang *et al*. [20] presents an energy-efficient architecture for IIoT. This architecture leverages RESTful service hosted networks, a cloud server, and user applications to build a hierarchical framework that balances the traffic load and enables a longer system lifetime. In addition a sleep scheduling and wake-up protocol is harnessed to predict sleep intervals, thereby promoting energy-efficiency at the system level.

A cloud-assisted IIoT–enabled framework for health monitoring is presented in [21] under the Healthcare Industrial IoT (HealthIIoT) theme. This framework leverages on sensors and mobile devices to collect and securely send health related data to the cloud for seamless access by healthcare professionals. As use case, authors deployed an IoT-driven ECG-based health monitoring service in the cloud. To prevent errors and misappropriation of data and information, authors proposed the use of signal enhancement, watermarking, and other related analytics.

B. SDN-BASED SOLUTIONS FOR IIoT

SDN was initially intended for traditional ICT networks, however recent innovations and advancement in SDN controllers enables SDN to swiftly integrate with IoT devices using special plugins on the southbound APIs. Most common among such controllers is OpenDaylight (ODL[1]); an OpenFlow-based SDN controller. ODL is the most commonly used SDN controller in both research and the industry. It supports over a dozen southbound APIs and protocols including Open vSwitch Database Management Protocol (OVSDB), Network Configuration Protocol (NetConf) and Path Computation Element (PCE) Communication Protocol (PCEP). These protocols are used to configure and manage network devices. The ODL reference architecture is presented in Fig. 2.

A plugin called IoTDM[2] is a dedicated IoT plugin specially designed to manage and store data generated by IoT device according to oneM2M[3] standard [23]. This IoTDM plugin is able to integrate to ODL southbound API and transfer data in

[1]https://www.opendaylight.org/
[2]https://github.com/opendaylight/iotdm/
[3]http://www.onem2m.org

both directions between the southbound and northbound. The oneM2M provides a standardized interface for managing and interacting with user applications. In [23], authors proposed a software architecture based on the oneM2M standard and using a system of plugins to ensure proper management of user applications. This proposed software architecture offers a modular platform based on pre-existing open-source software. This would serve as a workable solution to the current challenges of adopting industry 4.0 in medium and high-end factories. In addition, being a modular solution, it becomes easy for developers to build new industrial IoT plugins that can integrate future technologies to the IIoT system.

C. ORCHESTRATION TOOLS

Orchestration is a technology for controlling interactions between virtualized components such as containers and taking care of service composition, management, and termination. The most commonly used container orchestration technologies are Docker Swarm, Kubernetes and Mesos. These technologies provide automated support for functionalities like service discovery, load balancing, and software upgrades [24].

IIoT relies on heterogeneous devices working together, generating and sharing large amount of data via different communication protocols. This heterogeneity presents a major challenge to developers since applications and architectures would always need to be developed to meet the requirements of all underlying protocols. As such, there is need for such virtualization orchestration provided by Docker to enable distributed system deployments [25].

In [26], authors discussed the prospects of a joint edge and fog orchestration to cope with the vast data volume and low latency requirements of 5G and future networks. In this work, the role of such an orchestration platform is analyzed for different diverse 5G scenarios with specific emphasis to IIoT, vehicular communication, multi-access network integration, and localized real-time control. The efficient operation of resource-constrained devices in 5G, which would be mostly IoT devices, is one of the cardinal opportunities presented by the integration of such edge-and-fog orchestration capabilities to 5G. These resource-constrained IoT devices can then rely on the edge resources to execute some of their computationally and power demanding tasks, hence enabling a low-cost design for IoT devices without compromising their needed intelligence and capabilities.

In [27], authors discussed application orchestration in mobile edge cloud. Here the focus is on the benefits of mobile edge computing towards IoT deployment and how orchestration and application life cycle management plays out in MEC. Placement of components across several layers of telco network was one of the significant complexities discussed in this report, with regards to the implementation of the model. Other complexities such as computational complexity and inherent stochastic nature of the arising problems were also discussed. Mathematical modeling with constrained multi-objective optimization was presented to provide some simplification for implementing this model in a real-life scenario.

In [28] and [29], authors discussed different mechanisms that can be deployed to ensure scalability for a group of low mobility Machine-Type-Communications (MTC) devices at the edge of the RAN. These mechanisms are based on group profiling to reduce the amount of signaling and their contents, hence enhancing computation offloading and resource allocation for low-power IoT edge devices. With MEC as one of its key components, such mechanisms can speed-up data delivery with fewer requirements on the network capacity [30]. The work presented in [31] proposed a smart gateway solution for filtering IoT communications through some form of data trimming to reduce unnecessary communication that could burden the core network and the cloud datacenter. Thus, making the integration of IoT and cloud computing termed Cloud of Things (CoT) a more practical means of reducing the computational resources and data management needed on different IoT nodes.

Another work of interest was presented in [32], here an edge IoT architecture called edgeIoT was proposed to handle data streams at the mobile edge to address scalability problems with traditional IoT architectures. Here, instead of transmitting data streams generated from IoT devices to a remote cloud server for analysis, each Base Station (BS) is connected to a fog node, and the fog node provides computing resources locally. On top of the fog, nodes would be an SDN-based cellular core designed to facilitate packet forwarding among the fog nodes. Also, a hierarchical fog computing architecture is used on each fog node to provide flexible IoT services without compromising users privacy. This is accomplished by associating each user IoT node with a proxy virtual machine in the cloud to perform data computation and analysis before sending the metadata to the corresponding IoT application virtual machine for a response.

D. CUSTOM SOLUTIONS FOR CONSTRAINED SCENARIOS

With more emphasis on lean and efficient operations towards sustainable manufacturing, most works in IIoT solutions, just like in IoT are aimed towards lean and efficient designs, especially for constrained scenarios.

In [33], authors introduced *unikernels*, which are are single-purpose appliances that are specialised into standalone kernels, and sealed against modification when deployed to a cloud platform. Leveraging these unikernels will significantly reduce both computational and storage requirements in constrained scenarios, and at the same time improve efficiency and security while reducing the overall operational cost of the provided solution. In [34], authors demonstrate that unikernels do not actually require a virtual hardware abstraction, but can achieve similar levels of isolation when running as processes. Here unikernels serve to enable the reuse of VM isolation while also being lightweight. They eliminate the general purpose OS (e.g., Linux) from the VM and run applications directly on the virtual hardware. On the technical side, unikernels are able to offer an order

of magnitude reduction in code size without significant performance penalty. Thus making unikernels a go to solution for constrained IIoT application areas. Unikraft[4] is a common example of a unikernel designed for tailoring the operating system, libraries and tools to the particular needs of your application. This significantly cuts down on the size of both the container and the virtual machine, hence providing more efficient solution and also reducing the attack surface for the software stack.

uKontainer[5] is another custom solution for constrained application scenarios. Just like the unikernels, uKontainer is a lean container runtime solution which is aimed at providing industrial-standard container runtime using a simplified, robust and portable approach. Leveraging uKontainer in conjunction with unikernels for designing custom IIoT solutions will offer a significantly more optimized and efficient solution for constrained scenarios.

E. MEC-ENABLED IIoT DEVICES

The fundamental value proposition of MEC is to advance low latency, higher bandwidth, and more computational capabilities at the edge of mobile networks closer to the end users. This is the same location where IoT devices are expected to dominate in 5G and future networks. For developers and equipment providers, this opens up a vast potential for innovation towards the IoT applications. For service and content providers, this innovation introduces the need for more flexible and robust orchestration platforms that would coordinate these myriads of devices for resource allocation, data sharing, and service distribution.

Edge computing in general has been widely researched to optimize end-user device resources: cyberforaging, [35], grid technology [36], computation offloading [37], cloudlets [38], and fog computing [39]. Fog computing is mainly aimed at IoT applications that leverage a platform set that collectively assists UEs, while MEC is built on the premise of application-related enhancements with regards to feedback mechanisms, content processing, and information storage [30]. Fog computing extends cloud computing capabilities by moving computation and data storage to the edge of the network, allowing for reduced latency and response delay jitter for applications [40], [41].

These features are particularly critical for latency-sensitive applications such as AR, VR, gaming and video streaming, particularly, in an IoT environment where applications and sensor embedded physical devices can be leveraged as fundamental appliances and composed in a mashup style to control development cost and maintenance pressure [42]. With present-day design, there is a common tendency for IoT devices to experience crashes and timing failures from low-sensor battery power, high network latency, and low computational capabilities.

In all ramifications, orchestration remains the key concept within such distributed systems. It enables the alignment of deployed applications within the business interest of users. However, orchestration, as it is today, is unlikely to serve the needs of future IoT applications, mainly because of the diversity, e.g., configuration, location, reliability, scalability, and security that exists among IoT nodes [42].

F. CONTAINER-BASED MICROSERVICE ARCHITECTURES

In recent years, cloud services have been transforming from monolithic architectures towards microservice architectures, where services are composed of various microservices taking care of some limited set of functions [43], [44]. This microservice approach brings several benefits over monolithic architectures, including better maintainability, flexibility, scalability, efficiency, as well as reduced complexity. Since each microservice can be developed, tested, deployed, scaled, operated, and upgraded independently, the microservice model is also very flexible in the geographical distribution of computational tasks.

MEC brings new computational tier to cloud computing, between the datacenter and local devices [45]. By moving some functions from the datacenter to MEC, cloud systems can better serve applications requiring low latency while saving computational and networking resources at core networks and datacenters. MEC and microservice architecture fit well together: low latency and data processing services, e.g., filtering and fusions, are beneficial to deploy at MEC. Regarding latency, the roundtrip time between the local and MEC node is low, and for data, less of it needs to be delivered to the public cloud.

Microservice architectures are typically implemented using container technology [46]. Unlike the monolithic architectures where the whole system runs inside a single container. Here containers enable developing applications in a manner where only one or few processes run inside a single container. Docker containers provide a lightweight, low overhead and fast technology empowering the usage of microservice architectures [24], [47].

In general, the dynamic service deployment following mobile UEs can be realized using microservice architecture implemented using containers and their dynamical orchestration. In this scenario, the orchestrator would deploy the service instances in optimal locations in the MEC host based on available resources and minimum latency. This location would also need to be targeted close to the UE as they move across the RAN. The orchestrator is then used to reschedule resources and migrate the applications while simultaneously integrating with various parts of the RAN.

III. SYSTEM FRAMEWORK

Here we show a pilot to test the performance and resource utilization of an open source SDN-based solution for IIoT applications leveraging on virtualized systems. As a use case, we demonstrate a reactive MEC service migration by presenting a video streaming application. Here, a MEC server managed by Kubernetes applies a gray-scale filter on an incoming video. A UE captures colorized video stream and runs a web

[4]http://www.unikraft.org/
[5]https://github.com/ukontainer

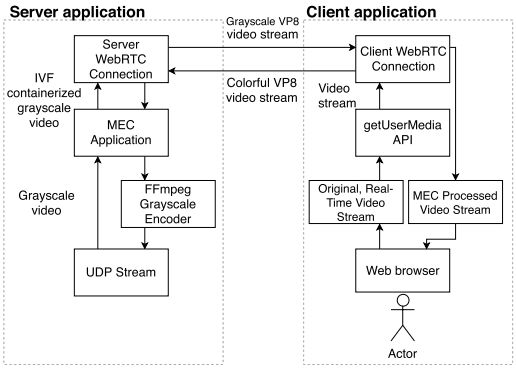

FIGURE 3. Flowchart of the MEC application logic.

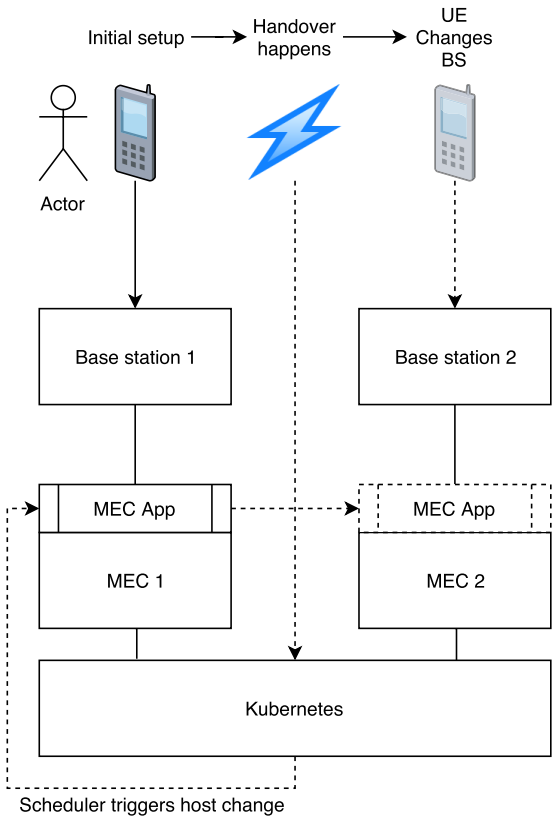

FIGURE 4. Reactive application migration logic.

application displaying its original colorized camera stream side-by-side with the MEC post-processed one. This setup easily reveals the latency resulting from the video processing and the network overheads. We then record MEC's end-to-end (E2E) latency, jitter, and service disruptions in real-time. We present data sets and the source code for the demos and integrations accessible on Github.[6]

Our environment, demonstrated in Figure 4, has two BSs and two MEC servers. First, the mobile device is assigned to a BS and have the MEC application running on MEC 1. Then, we move closer to the second BS, which will cause an S1 handover request, perceivable on the Evolved Packet Core (EPC). Now, the EPC uses the S1 handover request to reactively trigger a Kubernetes service migration to move the edge application from MEC 1 to MEC 2. The migration disrupts the UE application, perceivable by the grayscale video stream stopping and disconnecting. Now, the UE starts polling the MEC cluster to re-establish the video stream. Once Kubernetes has migrated and started the MEC application on the new host, a Domain Name System (DNS) update occurs. This DNS update points the old Multi-Access Edge Computing (MEC) service address, used by the UE application, to point to the new host. Now, the connection is re-established, and the grayscale video stream continues. This also marks our demonstration of a reactive MEC service application migration to end.

Our application is an extension to our previous research [48] in which an open-source RAN with Development Operations (DevOps) capabilities is presented. Here, the EPC is part of the same layer of the three network fabric as the MEC hosts through `flannel`. Because the EPC handles UE traffic, it can route requests between its intranetwork Kubernetes services and the UEs. As a new contribution, we introduce (1) an application in the environment, and (2) We use Kubernetes as an integrated facilitator for the MEC service migration on S1 radio handovers. We use S1 handovers instead of X2 because the EPC, which also runs the Mobility Management Entity (MME), can then tap network traffic to integrate MEC application migration via Kubernetes.

A. DESIGN OF THE CLIENT APPLICATION

To elaborate on the right side of Fig. 3, the client application uses web browser's `getUserMedia` Application Programming Interface (API) to access its live camera feed. We then leverage WebRTC to send a User Datagram Protocol (UDP) stream of the camera feed to the MEC. By relying on APIs available on web browsers, we achieve interoperability in our demonstration. That is, as long as the UE can run a modern web browser and has a camera attached to it, the UE can be used for the demo.

B. DESIGN OF THE SERVER APPLICATION

To elaborate on the left side of Fig. 3, the server application expects a VP8 encoded real-time transport protocol (RTP) stream over WebRTC from the client application. The server then encapsulates the data to an IVF container, and pipes it to `FFmpeg` [49]. `FFmpeg` is then responsible for applying a grayscale filter on the video, and emitting it as VP8 stream over UDP on a localhost socket. The server application then reads this local UDP socket and creates video samples to send them back to the client over WebRTC.

C. DESIGN OF REACTIVE SERVICE MIGRATION

As mentioned, we use Kubernetes as MEC orchestrator and an open-source NextEPC [50] as the EPC. To integrate MEC application migration on handovers, we create a script which scans the standard output of the EPC. Now, when the EPC logs a handover request, the script triggers an automatic service migration to the Kubernetes scheduler. Here, the original MEC application host is first cordoned as unschedulable.

[6]https://github.com/toldjuuso/handover2019okwuibe

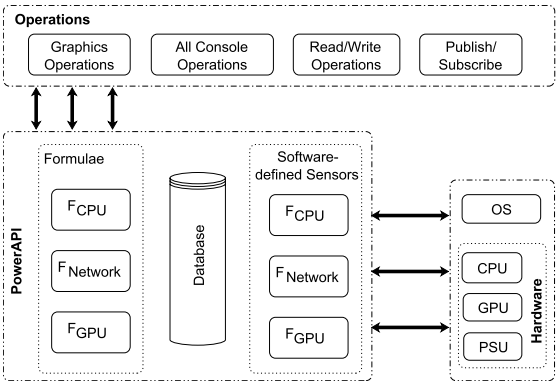

FIGURE 5. PowerAPI reference architecture.

Then the Kubernetes pod of the video post-processing service is killed. Kubernetes scheduler then reacts by placing the new pod on the only available server, which is MEC 2. Once the migration is done, CoreDNS [51], our chosen DNS server responsible for managing entries within the Kubernetes installation, updates the video post-processing service IP endpoint, thus making the service available again.

IV. POWER MEASUREMENT APPROACH AND DESIGN

To provide a detailed analysis on the power utilization and trade-offs for service orchestration in such containerized edge application, we propose a power measurement model that assesses power utilization on both the application and the connected node. At the application level, we proposed PowerAPI[7]; a software-defined power monitoring agent, while on the connected node, we used Monsoon power monitoring device.[8]

1) PowerAPI ARCHITECTURE

The PowerAPI is an open source platform for building software-defined power meters. As a software-defined solution, the PowerAPI and power meters are easily configurable sets of software libraries that are able to estimate the power utilization of applications and system processes in real-time. By simply collecting raw information from hardware devices through the operating system, they are able to provide fine-grained applications' power feedback information at runtime with an accuracy level comparable to, and in some cases, better than the traditional hardware-based approaches [52]. The PowerAPI architecture is highly modularized (see Fig. 5), which enables different modules to provide different measurement parameters from predefined components e.g. CPU, memory, disk.

More specifically, the PowerAPI models the CPU power according to standard CMOS[9] equation:

$$Power_{CPU}^{f,v} = c \times f \times V^2 \tag{1}$$

Here f is the frequency of the CPU, V is its voltage, and c is a constant value that is dependent on the properties of the hardware material like capacitance and activity factor [53]. It is important to note that power consumption does not always correlate with the percentage of CPU utilization, this is mainly due to the effect of Dynamic Voltage and Frequency Scaling factor (DVFS). For instance, a process with 70% CPU utilization and low voltage could be taking less overall power than a process with 50% CPU utilization but with a higher voltage. This explains why we cannot simply depend on the percentage of CPU utilization alone in estimating the power consumption of processes, hence our power model allows for much more accurate power consumption monitoring.

For our particular use case, where the overall power consumption at the application level is based on the CPU power model for software processes, this model is defined by the equation:

$$P_{comp} = \frac{\sum_{f \in frequencies} P_{comp}^{f} \times t_{CPU}^{f}}{\sum_{f \in frequencies} t_{CPU}^{f}} \tag{2}$$

which is the average power of the CPU for each frequency expressed as a ratio of the CPU time of all frequencies. While for a particular method in an individual application, we express the CPU power for such particular method in relation to the time as:

$$Power_{method}^{CPU} = \frac{Time_{method}^{CPU} \times Power_{thread}^{CPU}}{Duration_{cycle}} \tag{3}$$

where $Duration_{cycle}$ is the duration of the cycle being monitored. For the disk running the operating system, we model power utilization using the equation:

$$Power_{process}^{disk} = Bytes_{read} \times Power_{reading} + Bytes_{write} \times Power_{writing} \tag{4}$$

where $Byte_{read/write}$ is the amount of data in bytes that the process has read and written to the disk while $Power_{reading/writing}$ is the power per byte required to read or write from or to the disk. This is normally hardware dependent, as such the value is provided by the equipment manufacturer. Hence for a given method, the consumed power is related to the amount of data exchanged, this relationship is defined by the equation:

$$Power_{method}^{disk} = \frac{Bytes_{method}^{disk} \times Power_{process}^{disk}}{Bytes_{process}^{disk}} \tag{5}$$

At the network level, power consumption is estimated in a similar way to that of the CPU according to the model:

$$Power_{process}^{network} = \frac{\sum_{i \in states} t_i \times P_i \times d}{t_{total}} \tag{6}$$

where P_{state} is the amount of power consumed by the Network Interface card (NIC) in the state i as provided by the card manufacturer, d is the duration of the monitoring cycle while t_{total} is the total time during which data was transmitted through the NIC. Hence, the network power is calculated in relation to the number of transmitted bytes using the equation:

$$Power_{method}^{network} = \frac{Bytes_{method} \times Power_{process}^{network}}{Bytes_{process}} \tag{7}$$

[7] https://powerapi-ng.github.io/index.html
[8] https://www.msoon.com/ LabEquipment/PowerMonitor/
[9] Complementary Metal Oxide Semiconductor

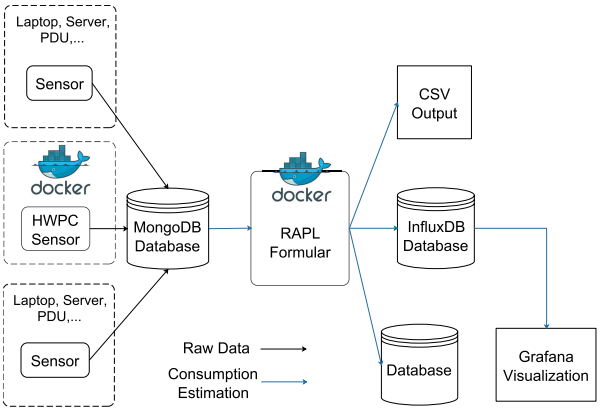

FIGURE 6. Software-defined power meter architecture.

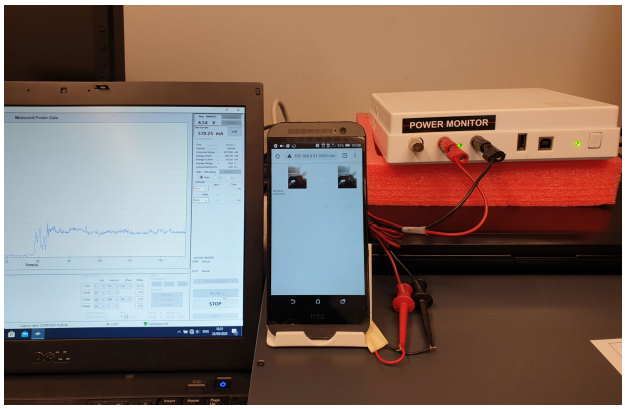

FIGURE 7. Power measurement on mobile node using Monsoon Power Tool.

where $Byte_{method}$ is the amount of R/W bytes performed by the method. $Power_{process}^{network}$ is the amount of power the application used while $Bytes_{process}$ is the amount of R/W bytes performed by all the methods in the application. As such, the power consumption per thread on the network is the sum of the power consumed by all methods running on the thread, i.e.

$$Power_{thread}^{network} = \sum Power_{methods}^{network} \tag{8}$$

2) SOFTWARE-DEFINED POWER METER

Also called power meter[10]; this is a simple software application built with the PowerAPI components, which is capable of measuring the power consumption of software processes running on a given machine or on cluster of machines. The power meter mainly consists of two elements: a software-defined sensor and an a formula, both working together to provide power consumption estimation for a monitored process or device. The sensor is an independent software that collects raw data correlated with the power consumption of monitored process while the formula also an independent software, uses the models presented in equations 1 - 8 to compute the power consumption of the monitored process or component from the data collected by the sensor.

Fig. 6 shows the architecture of the power meter we used in our implementation. Here, we used HWPC[11] and RAPL[12] technology as our sensor and formular respectively.[13] We used MongoDB[14] as the database to collect our raw sensor output. Hence the sensor writes the collected data to the database and the formula reads this data from the database. We connected our HWPC sensor and formula via MongoDB database in `Stream Mode`, this would enable us obtain real-time readings from the system.

For visualization, we used Grafana[15]; a Go-based open source multi-platform visualization web application. From Grafana we were also able to export our data directly to CSV for further analysis using MATLAB. To be able to visualize our data on Grafana, we needed an additional database to collect the data produced by applying the RAPL formula to the sensor data and connecting with Grafana. For that, we used InfluxDB database, an open source time series database developed by InfluxData[16] at MIT. InfluxDB is also based on Go language, it offers a cloud-based storage and retrieval capability with high-availability for monitoring IoT data and other real-time analytics.

3) MONSOON POWER MONITOR TOOL

The Monsoon power monitor is a Low Voltage Power Monitor (LVPM) capable of supplying and simultaneously measuring voltages between 2.1V - 4.55V DC. The maximum current draw and sampling rate are 3.0A and 200MHz respectively. On the one hand, these specifications make the Monsoon power monitor the perfect measurement tool for our mobile node, which is a HTC One (M8) mobile phone, but on the other hand we had a bottleneck because the Monsoon power monitor is not capable of measuring the power utilization on the server node, which is a Hp ProBook 6360b laptop computer, hence the need for the PowerAPI approach. Fig. 7 shows the setup of the power measurement on mobile node using the Monsoon power monitor.

Besides these measurement limits, the Monsoon tool provides fine-grained measurement data at different levels of details with various presentation tools to make for easy interpretation of collected data. A complementary software and a Graphical User Interface (GUI) provide a good number of adjustment and fine-tuning features for the Monsoon tool. For our use case, we exported collected data in CSV formats and analyzed using MATLAB.

Tables II and III show the hardware and software components of our testbed, respectively.

4) TESTBED SETUP

Our testbed is set to emulate an industrial application scenario where a machine vision system is harnessed for the

[10]https://powerapi-ng.github.io/powerapi_howitworks.html
[11]Hardware Performance Counters Sensor
[12]Running Average Power Limit
[13]https://powerapi-ng.github.io/hwpc.html
[14]https://www.mongodb.com/
[15]https://grafana.com/
[16]https://www.influxdata.com/products/influxdb-cloud/

TABLE 2. Hardware components used.

Component	Specifications
HP 6600-24G 4XG OpenFlow (OF) Switch - SDN switch	Gigabit Ethernet, PowerPC 8540 processor, 256 MB - SDRAM, 4 MB flash, RIP-1, RIP-2, static IP routing, IGMPv3, 666 MHz clock speed, 24 x 10/100/1000 + 4 x shared SFP + 4 x SFP+, RADIUS, Secure Shell v.2 (SSH2), TACACS+.
HP ProBook 6360b - Running mininet network emulator	Intel® Core™ i5-2540M dual-core (4 threads) processor (Sandy Bridge architecture). Speed: 2.60 GHz with Turbo Boost up to 3.3 GHz, TDP: 35W, Cache: 3 MB L3, 120 GB SSD. Ubuntu 18.04.5 LTS.
HP ProBook 6560b - Running Docker, PowerAPI, HWPC, MongoDB, RAPL, InfluxDB, Grafana, Go, FFmpeg	Intel® Core™ i5-2310M dual-core (4 threads) processor (Sandy Bridge architecture). Speed: 2.10 GHz, TDP: 35W, Cache: 3 MB L3, 120 GB SSD. Ubuntu 18.04.5 LTS.
Juplink RX4-1500 WiFi 6 Router	802.11ax/Wi-Fi 6, 300 Mbps at 2.4 GHz, 1200 Mbps at 5 GHz, 1.5 GHz, 1 Gigabit WAN port, 64-bit tri-core processor, OFDMA traffic management, WPA3 encryption.
Dell OptiPlex 990 SFF - Running Open Daylight SDN Controller	Intel® Core™ i3-2120 dual-core (4 threads) processor @3.30 GHz, Intel Q67 Express Chipset, HD Graphics 2000, Integrated Intel 82579LM Ethernet LAN 10/100/1000. 160GB HDD, Ubuntu 18.04.5 LTS.
HTC One M8 - Serving as IIoT node	Android 6.0. Chipset: Qualcomm MSM8974AB Snapdragon 801. CPU: Quad-core 2.3 GHz. GPU: Adreno 330. Selfie camera: Single 5 MP, f/2.0, HDR, Video 1080p@30fps, Wi-Fi 802.11 a/b/g/n/ac, dual-band.
Dell Latitude E6410 - Running PowerTool monitoring application	Intel® Core™ M460 dual-core processor @2.53 GHz, Mobile Intel HM57 Express, 8 GB DDR3 SDRAM, 120 GB SSD, Windows 10 Home 1909.
Monsoon Power Monitor - Monitoring power consumption on IIoT node	FTA22D, 6 VDC/5 A power supply, supplies and measures 2.1 V to 4.5 V in 0.01 V increments. Can support 3.0 A of continuous current and 4.5 A of peak current.

TABLE 3. Software tools used.

Tool	Parameters
OpenDaylight	Version Oxygen 0.8.4, OF SDN Controller
PowerTool	Version 5.0.0.25, GUI Software for Monsoon Power Tool
Docker	Version 19.03.6, orchestrator (native)
PowerAPI	Version 0.7.3, middleware toolkit for building software-defined power meters
HWPC	Software-defined sensor (Docker container)
MongoDB	Version 4.4.1. Database to collect sensor data, connects to RAPL formula (native)
RAPL	Technology for monitoring CPU power consumption (Docker container)
InfluxDB	Database to collect processed data from the RAPL formula, connects to Grafana (native)
Grafana	Version 7.2.0. Open source edition. Interactive real-time web visualization tool
Go	Version 1.12 (native)
FFmpeg	For video streaming (native/Docker)
Mininet	Version 2.1.0 (native)

monitoring of industrial processes. We model a 3-D drone system used in construction and car manufacturing industries for surveying and inspection. Here, we are considering a constrained scenario where power and computational resources are limited on the mobile node. As such the mobile node is only capable of capturing the video stream and transmitting through the network interface to a server on the edge of the network for processing. The server, in turn, is able to send some processed information back to the mobile node, hence a server-client communication setup. We used an HTC One M8 Android phone without batteries as the mobile node. The only source of power for the phone in our setup is the Monsoon power monitor which supplies and measures consumed power simultaneously. The phone captures video feed using the camera and sends the feed for processing on a docker run FFmpeg sever located on the edge of the network. Connection is managed using SDN. We configured Open Daylight on a Dell OptiPlex 990 SFF and using Mininet we setup a network using the HP6600-24G4XG OpenFlow (OF) Switch and the Open Daylight controller. On the same Edge server where the video stream is being processed in a docker container, we configured a power measurement solution based on PowerAPI and a software defined power sensor, HWPC. At the same time, we set up another FFmpeg server to run natively on the server. We compared these two scenarios for power consumption and latency overheads. This setup is depicted in Figure 8.

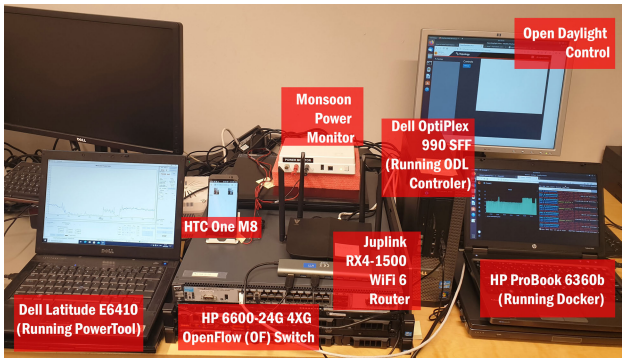

FIGURE 8. Overall testbed setup.

V. EVALUATION AND RESULTS

A. END-TO-END LATENCY FOR MOBILE NODE

The E2E latency was measured by holding a stopwatch in front of the UE camera on the client application. Now, we took screenshots of the UE application, which showed both the live camera feed side-by-side with the post-processed one. Thus, we could count the delta of the stopwatch times seen on the screenshot, resulting in E2E latency reading. We measured the end-to-end latency 95% percentile to be 1.06s, with a minimum and maximum delay of 0.09 s and 1.52s, and mean being 0.55s, as further demonstrated in Fig. 9.

B. ENERGY DRAW

Energy draw was measured using a power meter. The power draw was measured from Nvidia Jetson TX2 system on chip (SoC) device. The Jetson's hardware architecture is based on ARM; thus, the hardware is similar to those of mobile phones. We measured the Jetson drawing 2.1 watts on idle and 6.5 watts while encoding video. Here, we propose that

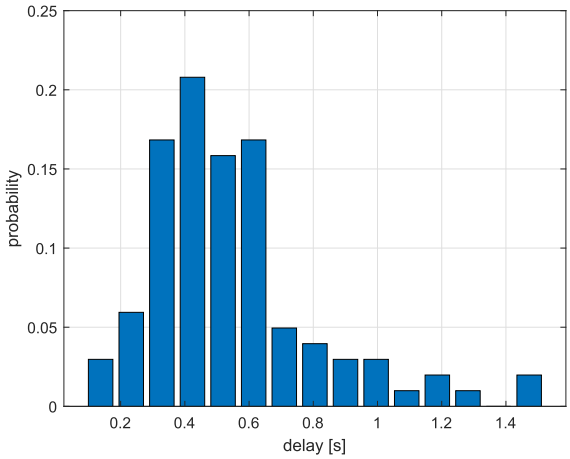

FIGURE 9. End-to-end latency readings (n = 100) visualized as histogram based on empirical cumulative distribution function.

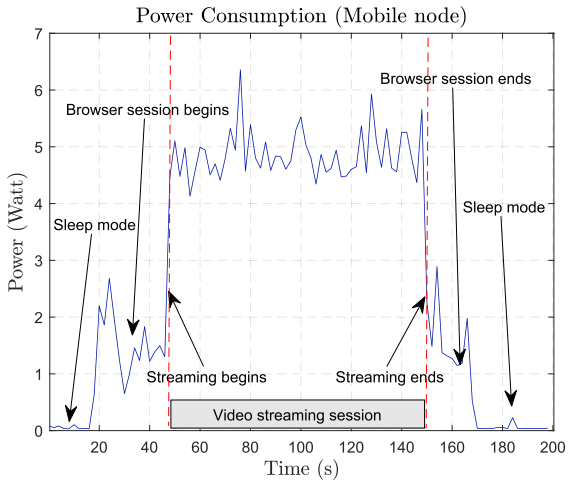

FIGURE 10. Power measurement for native implementation (mobile node).

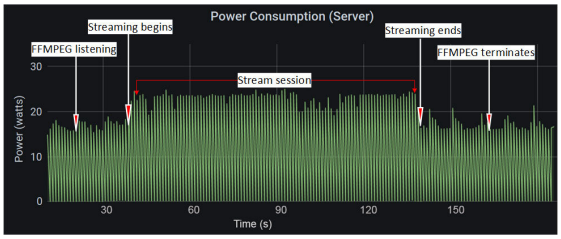

FIGURE 11. Power measurement for native implementation (server).

by offloading the video post-processing to the edge, we can reduce power consumption to 1/3 of what would be needed if the same workload would be done on-device.

C. APPLICATION MIGRATION LATENCY

Application migration latency was measured from Kubernetes logs. Here, the delta is counted between the time the pod was killed to when the pod self-reported it to be started from its logs. Hence, there are several convoluted factors that count into the migration latency, e.g., (1) the hardware performance of the Kubernetes scheduler, i.e., the primary host and the secondary nodes, (2) the network fabric, here, `flannel` used between Kubernetes primary host and the secondary nodes,

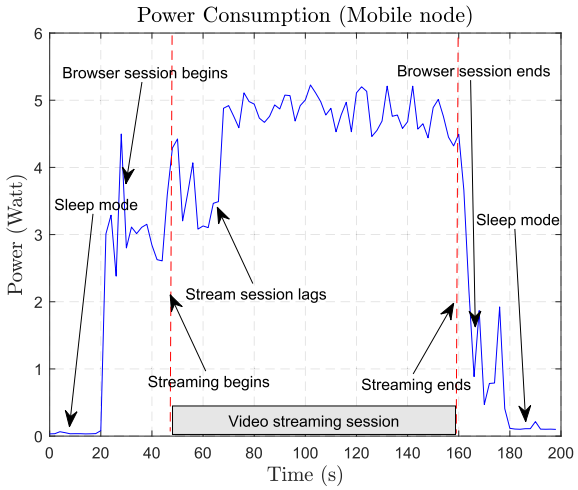

FIGURE 12. Power measurement for Docker implementation (mobile node).

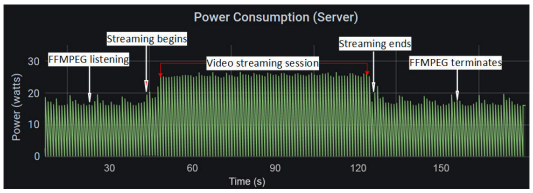

FIGURE 13. Power measurement for Docker implementation (server).

(3) the container virtualization method used, e.g., `Docker`, which we used, vs. `runc`, vs. `containerd`, vs. *et al.*, (4) the application initialization time, much dependent on programming language, here, Go. While many related works exist for dissecting the factors and approaches to minimize the latency in each category, we provide our benchmarks as a ballpark value. It is part of our on-going work to study approaches to reduce these latency factors towards current and future cellular ultra-reliable and low-latency communication (URLLC) networks. We measured the application migration latency 95% percentile to be 6.805s, with a minimum and maximum delay of 2.730s and 7.480s, and mean being 4.450s.

VI. DISCUSSION AND FUTURE WORK

In this work, we have investigated the trade-offs involved in leveraging software-defined solutions and container-based orchestration techniques in industrial automation. Quite evidently, introducing an intermediate layer to handle orchestration inevitably introduces some latency as our results have shown. However, with MEC integrated, these latencies could significantly be decreased.

We also considered how the proposed system would handle a handover event. For defining the handovers, we recorded the timestamp when the BS received an `End Marker` from the EPC. Radio Signal Strength (RSS) readings were measured using proprietary software called *Nemo Handy Handheld Measurement Solution* from Keysight Technologies Inc. The software was installed on a Samsung Galaxy S7 phone. Here, the software was used to record reference signals received

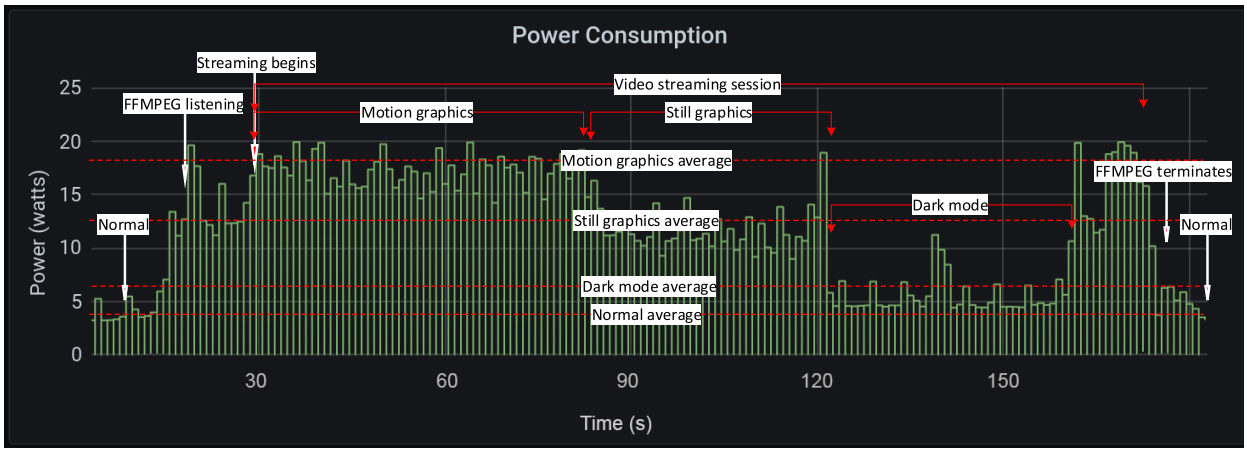

FIGURE 14. Power measurement for activity detection.

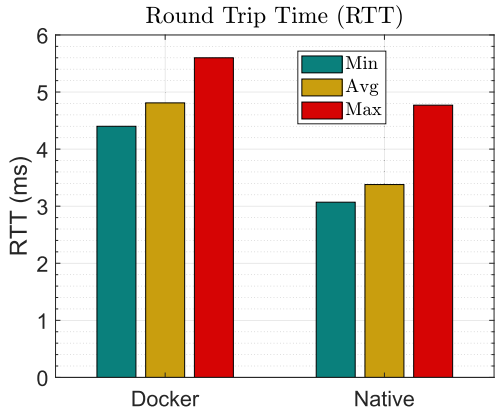

FIGURE 15. Latency comparison for native and Docker implementations (FFmpeg server).

power (RSRP) (Fig. 16) and received signal reference quality (RSRQ) (Fig. 17). The idea here was to provide datasets and reference values from which future work towards a pre-emptive MEC application migration model could base on. To elaborate, the MEC scheduler would use machine learning to migrate and initialize applications between MEC hosts ahead of time. This would result in a zero-downtime MEC application switch. Now, the MEC-dependent end-user application, such as the one proposed in Section III, would no longer be prone to downtime as measured in SectionV-C. We envision such a model being one practical approach towards addressing problems in cellular URLLC application such as autonomous cars, in which the system is at least partly dependent on information coming from the edge. We note that this model likely requires both the little varying RSRQ readings and the more varying RSRP readings.

Further, for a scalable system, these values should be collected from the BSs instead of individual UEs. However, proprietary BS software might make it close to impossible to apply these changes without the cooperation of the hardware providers. This problem could be addressed either by open-source BS software, or standardization, such as APIs, which facilitate software approaches leveraging RAN data analytics for intelligent radio resource optimization.

It is important to note that the overall benefit of orchestration in such IIoT applications goes beyond the mere ability to offload complex computational processes to more capable platforms, i.e. providing Platform-as-a-Service and energy savings. The impact of this change in paradigm on CapEx and Operating Expenses (OpEx) could be massive when applied on a large-scale industrial complexes. Our use case was a drone-based monitoring system, however, the approach in our prototype can be applied to other IIoT applications.

In any case, the potential for further optimization is still huge. All components we used in our prototype are basically all-purpose off-the-shelf components, as such, the energy saving potential from our demonstrations could be a fraction of what is attainable with custom components. For instance, in our demo, we used a basic Android phone to emulate the drone camera system. As much as we had the phone configured to basic factory settings, running a full Android OS and powering up a several other background processes and services obviously introduces unnecessary overheads. Our use case needed only a simple device with a camera, a browser, and a network interface. With such a lean design, the potential for prolonged remote operation of such an IIoT device is certain. The same applies to the software components we used. On the server side, we configured the PowerAPI and its software defined elements, as well as the Floodlight controller and FFmpeg server on the default Ubuntu kernel. However, some custom Linux systems called unikernels and custom container technologies such as uKontainer[17] could be leveraged for particular use cases, especially in constrained scenarios [54]. The uKontainer, for instance, is a recently developed industry-standard container runtime specially designed for simplicity, robustness and portability. Such solutions will further eliminate unnecessary overheads involved in running heavy kernels, hence enabling further resource optimization when used in such constrained applications. In our prototype, we noticed a substantial variation of about 7 watts when running the same exact Docker containers and local services on the HP ProBook 6360b with Intel® Core i5-2540M

[17]https://pkg.go.dev/mod/github.com/ukontainer/containerd

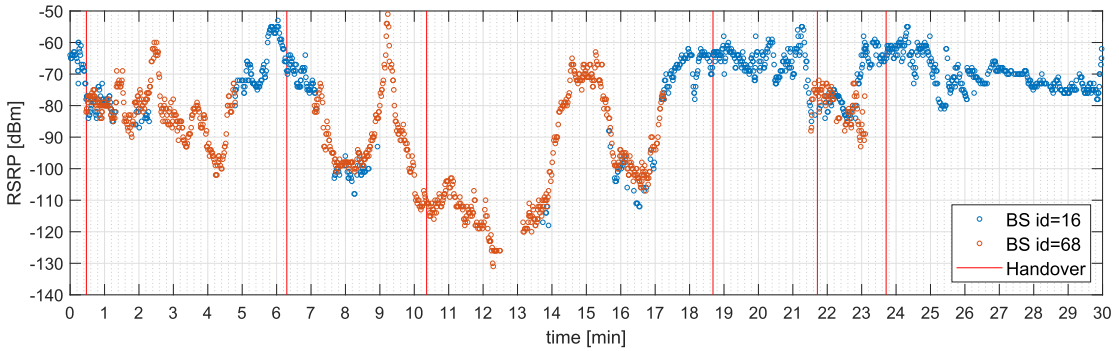

FIGURE 16. RSRP readings. Here, −70 or lower indicates an excellent signal, whereas −110 or less indicates little to no signal.

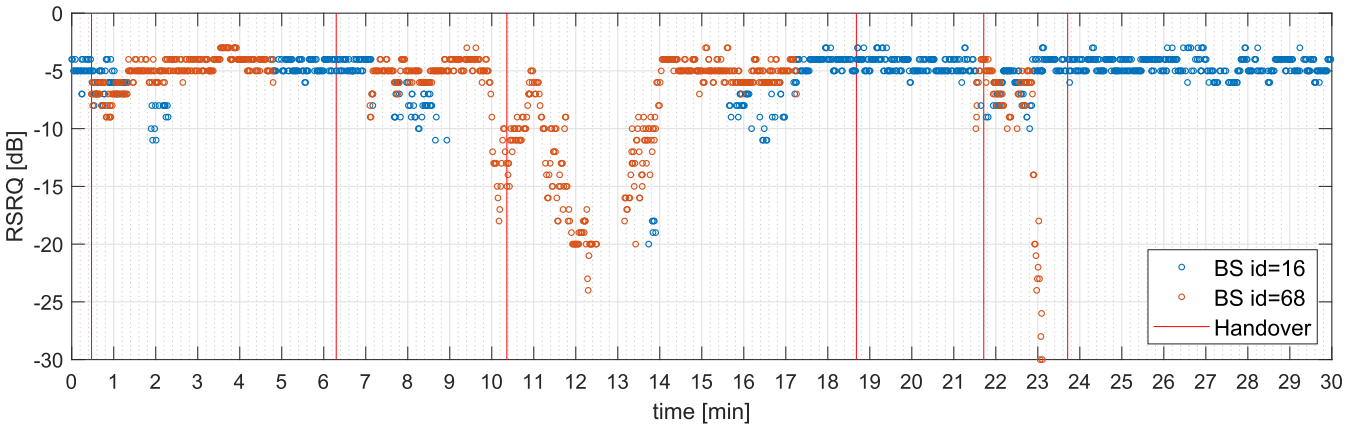

FIGURE 17. RSRQ readings. Here, quality of around −3 indicates an excellent signal, whereas −16 or lower as unusable one.

dual-core processor @ (2.60 GHz) compared to the HP ProBook 6560b with Intel® Core i5-2310M dual-core processor @ (2.10 GHz) both rated 35W TDP. With software-defined solutions, such as SDN and software-defined power meters and formulars, such optimization techniques become even easier to harness.

Another key element of the prototype was the visual monitoring potential for such a lean design solution. As shown in Fig. 14, this basic model is able to detect variations in video streams solely from variations in the real-time power required for the FFmpeg encoding. For instance light vs. dark colors, still vs. motion images. When fully developed, this element would potentially support applications in visual use cases such as drone thermography, modular cleaning management and other Supervisory Control and Data Acquisition (SCADA) systems.

In a car assembly line, for instance, such solution can be used to monitor robotic painting of cars in real-time and providing real-time paint thickness information via light reflection spectrum analysis and the real-time video encoding power spectrum through an automated feedback loop system. The same solution can also be applied to oil pipeline monitoring. In such use cases, our proposed solution is capable of interacting in real-time with industrial sensors, PLCs, and cloud automatically via an IoT gateway as a middleware and to transmit data between the different systems securely.

As future work, we would like to extend from orchestrating a single IIoT network to orchestrating multiple networks with latency critical/resource constrained IIoT devices. In addition, it will be interesting to further optimize the visual accuracy of the pilot system towards more advance visual AI use cases for SCADA systems and other industrial automation application areas.

VII. CONCLUSION

In this article, we studied how integrating SDN, MEC, and container-based orchestration would affect resource and performance optimization in IIoT applications. In particular, practical concerns of integrating an open-source container orchestrator system and the constraints under which UE might depend on MEC resources, were addressed.

First, we presented an MEC application which used MEC resources to post-process video coming from UE. The application was deployed as a Kubernetes service on an existing open-source RAN at the University of Oulu. The results related to using existing container orchestration systems, here, Kubernetes, for reactively moving MEC applications closer to UEs showed a mean migration time of 4.450s, this underscored the need for new approaches to application migration in future MEC-enabled cellular networks. To provide data for future research, we gathered the RSS readings from our base stations. With this information, machine learning could be used to pre-emptively move containers ahead-of-time, and as such, address the migration times.

Second, we analysed the power trade-offs involved in such an approach at both the application and IIoT mobile nodes.

For our use case, the mobile node only required a camera, a web browser and a network interface. Our measurement results showed that ∼16% of the total power consumption happened on the mobile node, which makes a good use case for orchestration. Our results also showed that the Docker visualization adds up ∼4.5% to the power consumption on the server node compared to local implementation. However, there was ∼5 seconds lag for the initialization of the streaming session on the mobile node when the FFmpeg server is running in a Docker compared to the native implementation. Based on this outcome, we are convinced that combining SDN, edge, and container orchestration in the manner we proposed here would, indeed, provide a feasible solution for IIoT applications without severe constraints. We also believe that the fine-grained data gathered from the PowerAPI was sufficient to provide information about the visual variations in the streamed images, thereby paving way for a more advanced use cases in visual AI applications.

REFERENCES

[1] T. Lins, R. Augusto Rabelo Oliveira, L. H. A. Correia, and J. Sa Silva, "Industry 4.0 retrofitting," in *Proc. VIII Brazilian Symp. Comput. Syst. Eng. (SBESC)*, Nov. 2018, pp. 8–15.

[2] K. Ahmed, J. Blech, M. Gregory, and H. Schmidt, "Software defined networks in industrial automation," *J. Sensor Actuator Netw.*, vol. 7, no. 3, p. 33, Aug. 2018.

[3] (2019). *Beginners: What is Industrial IoT (IIoT)*. [Online]. Available: https://www.3g4g.co.uk/IoT/iotm2m_0004.html

[4] J. Okwuibe, M. Liyanage, and M. Ylianttila, "Provider assisted Wi-Fi offloading leveraging on SDN," in *Proc. 22th Eur. Wireless Conf.*, May 2016, pp. 1–6.

[5] T. Skeie, S. Johannessen, and O. Holmeide, "Timeliness of real-time IP communication in switched industrial Ethernet networks," *IEEE Trans. Ind. Informat.*, vol. 2, no. 1, pp. 25–39, Feb. 2006.

[6] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," *Proc. IEEE*, vol. 93, no. 6, pp. 1102–1117, Jun. 2005.

[7] C. Rojas and P. Morell, "Guidelines for industrial Ethernet infrastructure implementation: A control engineer's guide," in *Proc. IEEE-IAS/PCA 52nd Cement Ind. Tech. Conf.*, Mar. 2010, pp. 1–18.

[8] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 4258–4265, Oct. 2009.

[9] L. Hou and N. W. Bergmann, "Novel industrial wireless sensor networks for machine condition monitoring and fault diagnosis," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 10, pp. 2787–2798, Oct. 2012.

[10] R. Zhang, Y. Chen, B. Dong, F. Tian, and Q. Zheng, "A genetic algorithm-based energy-efficient container placement strategy in CaaS," *IEEE Access*, vol. 7, pp. 121360–121373, 2019.

[11] (2017). *Exastax: Top 5 Benefits of Containerization*. [Online]. Available: https://www.exastax.com/containerization/top-5-benefits-of-containerization/

[12] J. Baten, *A History of the Global Economy*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[13] S. BMIET, "Industrial revolution–from industry 1.0 to industry 4.0," *J. Adv. Comput. Intell. Commun. Technol.*, vol. 2, no. 1, pp. 1–2, 2018.

[14] W. Rosen, *The Most Powerful Idea in the World: A Story of Steam, Industry, and Invention*. New York, NY, USA: Random House, 2010.

[15] M. S. Vassiliou, *Historical Dictionary of the Petroleum Industry*. Lanham, MD, USA: Rowman & Littlefield, 2018.

[16] J. H. Lienhard, *The Engines of Our Ingenuity: An Engineer Looks at Technology and Culture*. Oxford, U.K.: Oxford Univ. Press, 2003.

[17] H. Kagermann, W. Wahlster, and J. Helbig, "Secure the future of Germany as a production location, implementation recommendations for the future project industry 4.0," in *Original Citation: Deutschlands Zukunft als Produktionsstandort Sichern, Umsetzungsempfehlungen für das Zukunftsprojekt Industrie*, vol. 4. Berlin, Germany: Forschungsunion, 2012.

[18] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial Internet of Things (IIoT): An analysis framework," *Comput. Ind.*, vol. 101, pp. 1–12, Oct. 2018.

[19] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*. New York, NY, USA: Springer, 2016.

[20] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu, "Green industrial Internet of Things architecture: An energy-efficient perspective," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 48–54, Dec. 2016.

[21] M. S. Hossain and G. Muhammad, "Cloud-assisted Industrial Internet of Things (IIoT)–Enabled framework for health monitoring," *Comput. Netw.*, vol. 101, pp. 192–202, Jun. 2016.

[22] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.

[23] J. L. Romero-Gázquez and M. V. Bueno-Delgado, "Software architecture solution based on SDN for an industrial IoT scenario," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–12, Sep. 2018.

[24] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in *Proc. IEEE/ACM 1st Int. Workshop Establishing Community-Wide Infrastruct. Archit.-Based Softw. Eng. (ECASE)*, Piscataway, NJ, USA, May 2017, pp. 8–13, doi: 10.1109/ECASE.2017.4.

[25] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, "Orchestration of containerized microservices for IIoT using docker," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Mar. 2017, pp. 1532–1536.

[26] L. Cominardi, O. I. Abdullaziz, K. Antevski, S. B. Chundrigar, R. Gdowski, P.-H. Kuo, A. Mourad, L.-H. Yen, and A. Zabala, "Opportunities and challenges of joint edge and fog orchestration," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 344–349.

[27] A. Hegyi, H. Flinck, I. Ketyko, P. Kuure, C. Nemes, and L. Pinter, "Application orchestration in mobile edge cloud: placing of iot applications to the edge," in *Proc. IEEE 1st Int. Workshops Found. Appl. Self* Syst. (FAS*W)*, Sep. 2016, pp. 230–235.

[28] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[29] T. Taleb, A. Ksentini, and A. Kobbane, "Lightweight mobile core networks for machine type communications," *IEEE Access*, vol. 2, pp. 1128–1137, 2014.

[30] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[31] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Proc. Int. Conf. Future Internet Things Cloud*, Aug. 2014, pp. 464–470.

[32] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.

[33] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 461–472, 2013.

[34] D. Williams, R. Koller, M. Lucina, and N. Prakash, "Unikernels as processes," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 199–211.

[35] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 1232–1243, 4th Quart., 2012.

[36] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 3, pp. 200–222, Aug. 2001.

[37] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generat. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[38] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervas. Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[39] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, Nov. 2015, pp. 73–78.

[40] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 13–16.

[41] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, 2015, pp. 37–42.

[42] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for Internet of Things services," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 16–24, Mar. 2017.
[43] C. Esposito, A. Castiglione, and K.-K.-R. Choo, "Challenges in delivering software in the cloud as microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 10–14, Sep. 2016.
[44] M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy Web applications in the cloud," in *Proc. 10th Comput. Colombian Conf. (10CCC)*, Sep. 2015, pp. 583–590.
[45] A. Reznik, R. Arora, M. Cannon, L. Cominardi, W. Featherstone, R. Frazao, F. Giust, S. Kekki, A. Li, D. Sabella, C. Turyagyenda, and Z. Zheng, "Etsi white paper no. 20: Developing software for multi-access edge computing," ETSI, White Paper 20, Sep. 2017.
[46] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, "Performance evaluation of microservices architectures using containers," in *Proc. IEEE 14th Int. Symp. Netw. Comput. Appl.*, Sep. 2015, pp. 27–34.
[47] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, "Exploring container virtualization in IoT clouds," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, May 2016, pp. 1–6.
[48] J. Haavisto, M. Arif, L. Loven, T. Leppanen, and J. Riekki, "Open-source RANs in practice: An over-the-air deployment for 5G MEC," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Valencia, Spain, Jun. 2019, pp. 495–500.
[49] FFmpeg Team. (Jun. 2019). *Ffmpeg–A Complete, Cross-Platform Solution to Record, Convert and Stream Audio and Video*. [Online]. Available: https://ffmpeg.org.v4.1.3. [Online]. Available: https://ffmpeg.org
[50] L. Sukchan. (Aug. 2018). *NextEPC: An Open Source Implementation of the Evolved Packet Core of LTE Networks Supporting 3GPP Release 13*. [Online]. Available: https://open5gs.org/nextepc/release/2018/08/17/release-v0.3.10.html.v0.3.10.
[51] Cloud Native Computing Foundation. (Apr. 2019). *CoreDNS: DNS and Service Discovery*. [Online]. Available: https://coredns.io/v1.5.0. [Online]. Available: https://coredns.io/
[52] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "Runtime monitoring of software energy hotspots," in *Proc. 27th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2012, pp. 160–169.
[53] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Operating Syst. Rev.*, vol. 47, no. 3, pp. 42–49, Nov. 2013.
[54] T. Mekonnen, M. Komu, R. Morabito, T. Kauppinen, E. Harjula, T. Koskela, and M. Ylianttila, "Energy consumption analysis of edge orchestrated virtualized wireless multimedia sensor networks," *IEEE Access*, vol. 6, pp. 5090–5100, 2018.

JUDE OKWUIBE received the B.Sc. degree in telecommunications and wireless technologies from the American University of Nigeria, Yola, in 2011, and the master's degree in wireless communications engineering from the University of Oulu, Finland, in 2015. He is currently pursuing the Ph.D. degree in communications engineering with the University of Oulu Graduate School (UniOGS), Finland. His current research interests include software defined networking, 5G and future networks, edge computing, the Internet of Things, industrial internet of things, container orchestration, SDN, network security, and biometric verifications.

JUUSO HAAVISTO is currently pursuing the joint M.Sc. degree with the Université de Lorraine, France (formal reasoning), and the University of St Andrews, U.K., (software engineering). He is also an Erasmus Mundus Joint Master degree (EMJMD) full scholarship student with the University of St Andrews, studying Advanced Systems Dependability. He will continue to doctoral studies in fall 2021 to study total functional programming languages on single instruction, multiple data (SIMD) parallel computers. He has worked for startups in Silicon Valley, published two first-authored articles with practical artifacts on 5G edge computing, and operated a limited liability company since the age of 17, specializing in software contracting using Go and proof-of-concept development in IoT. Y Combinator, a startup incubator in Silicon Valley invited him in 2018 to on-site interviews (acceptance rate 4%) over Periferia, a project about developer tools for reducing 5G network end-to-end latency.

ERKKI HARJULA (Member, IEEE) received the M.Sc. degree in computer engineering and the D.Sc. degree in communications engineering from the University of Oulu, Finland, in 2007 and 2016, respectively. He is currently a full-time Assistant Professor (tenure track) with the Center for Wireless Communications (CWC), University of Oulu. Previously, he worked as a Postdoctoral Researcher and a Project Manager with the CWC research group from 2016 to 2020, and a Research Scientist with the Center for Internet Excellence (CIE) from 2013 to 2015, and MediaTeam research group from 2000 to 2014. He visited Columbia University during winter 2008–2009. He is the coauthor of over 65 international peer-reviewed scientific articles on mobile and IoT systems, edge computing, cloud computing, distributed systems, and green computing.

IJAZ AHMAD (Member, IEEE) received the M.Sc. and Ph.D. degrees in wireless communications from the University of Oulu, Oulu, Finland, in 2012 and 2018, respectively. He had been a Postdoctoral Fellow with the Center for Wireless Communications, Oulu, from 2018 to 2019. He had been a Visiting Scientist with Aalto University, Finland, in 2018. Since 2019, he has been working as a Research Scientist with the VTT Technical Research Center of Finland. In 2019, he visited TU Vienna, Austria, to work with Prof. T. Sauter as a Visiting Scientist. His research interests include 5G, 6G, 5G security, the IoT, and the application of machine learning in wireless networks. He was a recipient of several awards, including the Nokia foundation, Tauno Tönning and Jorma Ollila grant awards, and two IEEE best paper awards.

MIKA YLIANTTILA (Senior Member, IEEE) received the M.Sc., D.Sc., and executive master of business administration (eMBA) degrees, and the Ph.D. degree in communications engineering from the University of Oulu, Finland, in 2005. He is currently a full-time Associate Professor (tenure track) with the Center for Wireless Communications (CWC), Faculty of Information Technology and Electrical Engineering (ITEE), University of Oulu. He leads Network security and softwarization (NSOFT) research group, at CWC Networks and Systems research unit, which studies and develops secure, scalable, and resource-efficient techniques for 5G and beyond 5G systems. He is also the Director of communications engineering doctoral degree program. Previously, he was the Director of the Center for Internet Excellence from 2012 to 2015, the Vice Director of MediaTeam Oulu research group from 2009 to 2011, and a Professor (pro tem) of computer science and engineering from 2005 to 2010. He has coauthored more than 180 international peer-reviewed articles. His research interests include network security, edge computing, network virtualization, and software-defined networking. He is also an Editor of *Wireless Networks* and an Associate Editor of IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY.